The Taxi Oracle

Daniel Suarez Souto

Abhishek Singh

Anurag Gunti

Julen Arizaga Echebarria

Alexis Gaziello

INDEX

Introduction of Business Problem

Motive - Developing a model that could benefit a private transportation company by saving a lot of time searching for passengers.

In order to make the private company's taxis reach more customers and earn a good reputation among them, the goal is to minimize the waiting time of customers.

Other external factors that have to be considered to design the best model include weather and holidays.

- ❑ Extreme weather conditions affecting the customers' rapid service.
- ❑ Public holidays like Thanksgiving, Christmas etc. and weekends where the demand of taxis would be higher than usual.

Problem Statement :

- To create a model that will predict the demand of taxis per hour and location that a private transportation company will need to perform trips, at a given location and time.
- The performance of this model will be analyzed not just in terms of prediction error (MAE and MSE), but also in terms of financial profit.

Importance of Problem :

- Our model will be helpful in predicting the number of taxi trips per each pickup community area every hour in Chicago.
- By utilizing our model, the company will be able to know much in advance the communities of Chicago where there is demand for a greater number of taxis so that only taxi drivers in that area can respond
- The private company will be able to make more profit by saving the time taken by taxi drivers particularly by avoiding the long-distance pickups.
- Deploying the exact number of taxi drivers in the pickup community will avoid wasting resources of the company and result in overall good customer satisfaction.

Data Collection

There are 3 datasets found to be useful for the model :

1. **Taxi Trips**[1]

   Taxicabs in Chicago, Illinois, are operated by private companies and licensed by the city. There are about seven thousand licensed cabs operating within the city limits.

   This dataset includes all taxi trips from 2013 to present date, reported to the City of Chicago.

   The trip times are rounded to the nearest 15 minutes. Due to the data reporting process, not all trips are reported but the City believes that most are.Each taxi is represented by a unique identifier known as **taxi_id.**

   Each line of this file after the header row represents one trip taken by taxi, and has a total of 23 attributes.

   Handling Big data :- The size of the Chicago Taxi dataset is 70 GB. Because computation time would have been to long, and the price of dealing with that much data would have been too big, we used data from the year 2017 to 2019 and from the Flash cab company.

   To get the data we had to go through the Google Cloud platform Big Query[2] application. We could have used a classic SQL to save it as a csv file, however, by reading a little more, we to download only the taxi trips made between the

   The dataset was download using following query:-

```
SELECT
 *
FROM

`bigquery-public-data.chicago_taxi_trips.taxi_trips`

WHERE

trip_start_timestamp BETWEEN "2017-01-01 00:00:00 UTC" AND "2019-12-31 23:23:59 UTC"

ORDER BY

trip_start_timestamp DESC
```

2. **Weather**[3]

   The dataset consists of the temperature of chicago at every hour from Jan 2017 to till date. Along with wind speed and relative humidity.

   This dataset has a total of 34 variables.

3. **Holidays**[4]

1 Source: https://www.kaggle.com/chicago/chicago-taxi-trips-bq
2 Source: https://bigquery.cloud.google.com/dataset/bigquery-public-data:chicago_taxi_trips
3 Source: https://mesonet.agron.iastate.edu/request/download.phtml?network=IL_ASOS
4 Source: https://data.cityofchicago.org/Transportation/CTA-Ridership-Bus-Routes-Daily-Totals-by-Route/jyb9-n7fm/data

This dataset gives us information on the type of day we are analyzing:

- Week day, Saturday, Sunday or Holiday

Each line of this file after the header row represents a day and whether it is a working day or holiday, and has the following format : date, daytype, route, rides

Data Cleaning

Trips

Variable selection

The first thing we did was dropping out attributes that we were definitely not going to use. Those are:

- **company**: Since we are working with only one company, this attribute is the same for all the rows and useless for prediction.
- **pickup_latitude, pickup_longitude, pickup_location, dropoff_latitude, dropoff_longitude, dropoff_location**: due to privacy matters, most of these values were missing, and for our study, we decided to limit our target variable to the community area. For that reason, we didn't use the latitude/longitude variables.
- **pickup_census_tract, dropoff_census_tract**: another measure of location. Also had lots of missing values, so for the same reasons that we dropped the latitude, we dropped this variables.

Missing values

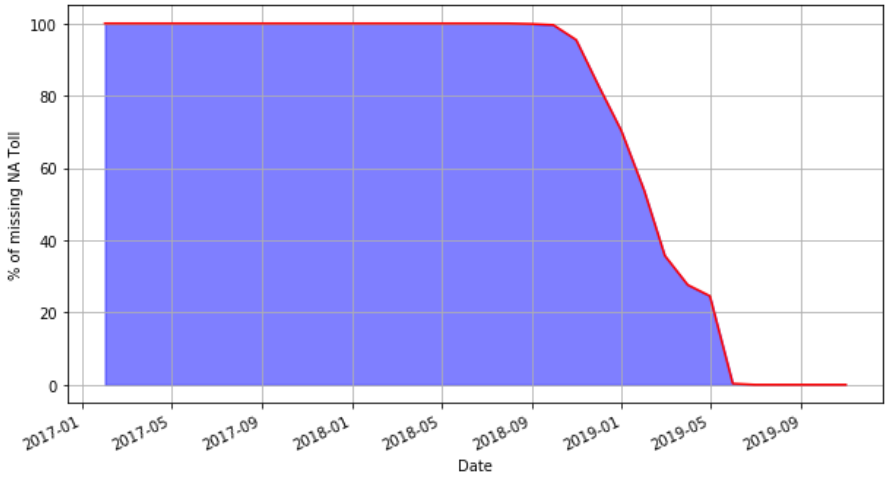
Initial state of dataset:

Variable	**Percentage of missing values**
unique_key	0.00000%
taxi_id	0.00000%
trip_start_timestamp	0.00000%
trip_end_timestamp	0.00000%
trip_seconds	0.00506%
trip_miles	0.00013%
pickup_community_area	0.00000%
dropoff_community_area	0.00000%
fare	0.00013%

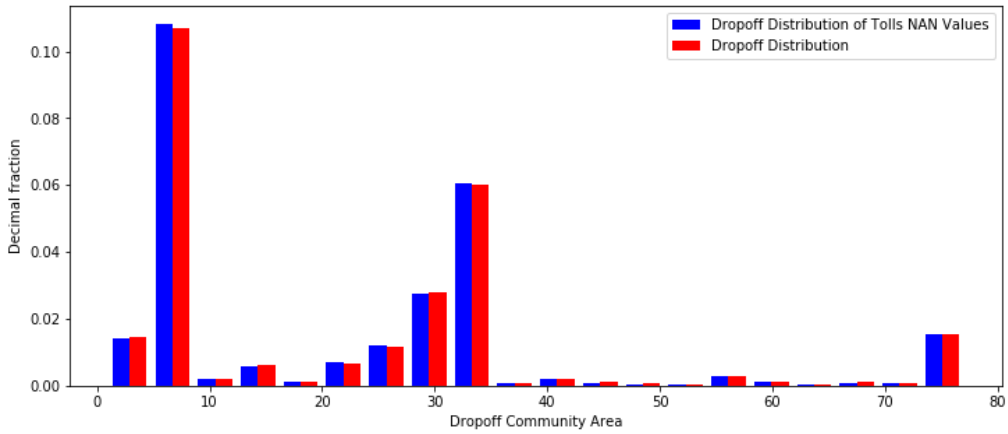
tips	0.00013%
tolls	77.49560%
extras	0.00013%
trip_total	0.00013%
payment_type	0.00000%

The first variable to spring to mind is clearly **tolls**, which has an enormous relative quantity of missing values. We analyzed this variable and we discovered a few things:

- It only has one possible value, when it's not missing. This value is 0.
- The variable was completely missing up until the end of 2018, when it slowly started being implemented:



- The distribution of missing values seems random:



After discovering these facts, we assumed that **tolls** was a new variable that was added recently to the dataset, indicating if the trips went through a toll or not. Since in Chicago there are not that many tolls, it seems that no taxi trips employed any of them.

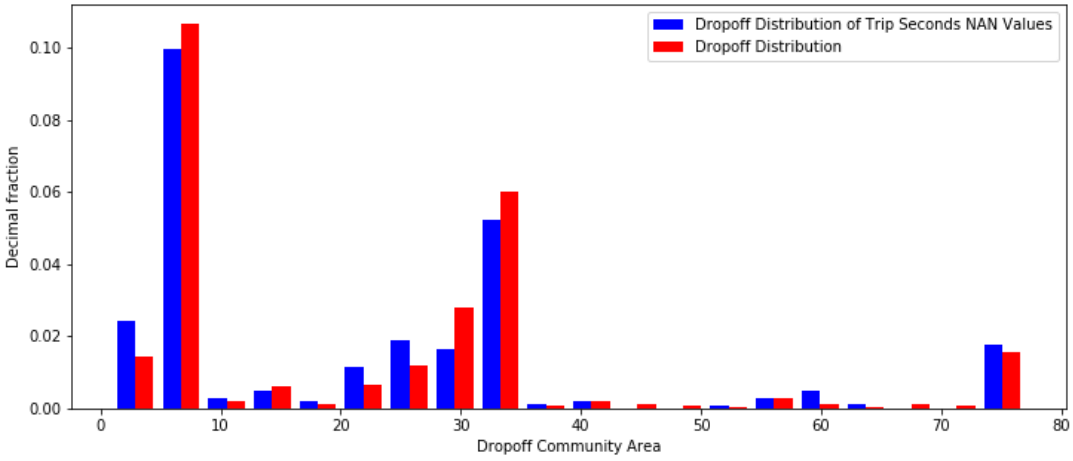
We concluded that this variable was useless for predicting or information so we dropped it.

trip_miles, fare, tips, extras & trip_total

These variables have the exact same amount of missing values, 0.00013%, which corresponds to 10 values. The exact same amount was curious and we thought that maybe the missing values belonged to the same rows. After checking that indeed the missing values corresponded to the same 10 rows, and since the percentage is so insignificant, we decided to drop them. These missing values were probably due to an error.

trips_seconds

Finally, **trips_seconds** seemed to have a few missing values too. After checking that the distribution of these missing values was random:



We decided to estimate them. **trips_seconds** represents the duration of the trip. We took the mean of the duration of the trips that started and ended in the same community area, and we filled the missing values with it.

Errors checking

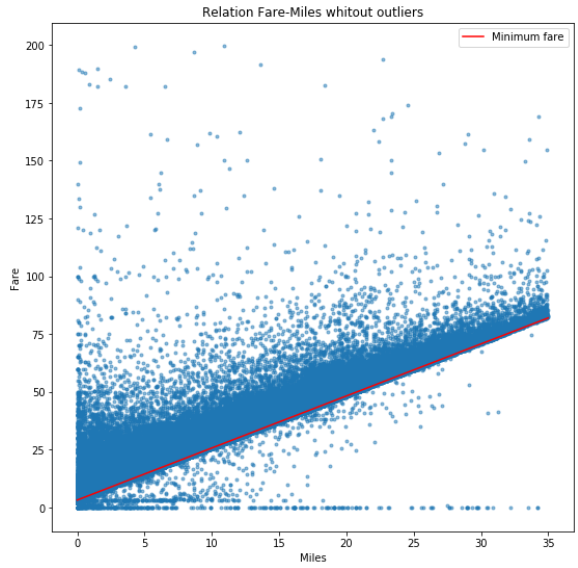
The first thing we did was checking the boxplots, but there were a lot of weird values and outliers, so we changed our strategy.

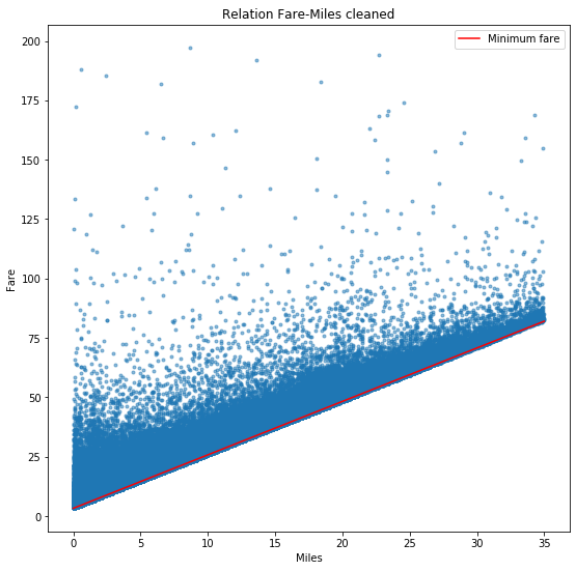
By looking a little in internet, we could find the amount that the company charged per trip[5]. The following equation represents the minimum amount, but other charges may be applied depending on the number of passengers or the destination.

fare = 3.25 + 2.25 per miles + 0.2 per 36 seconds waiting

With this equation, we could plot fare versus miles and time and discover errors.

[5] Rates: https://flashcab.com/rates/

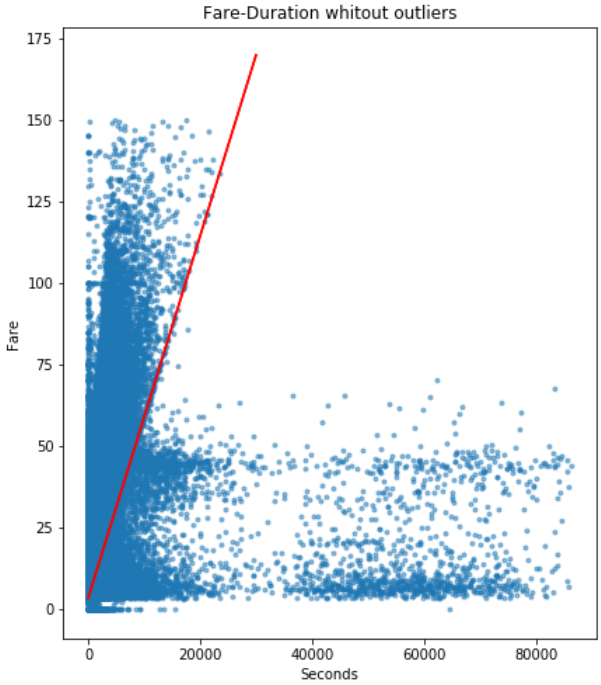


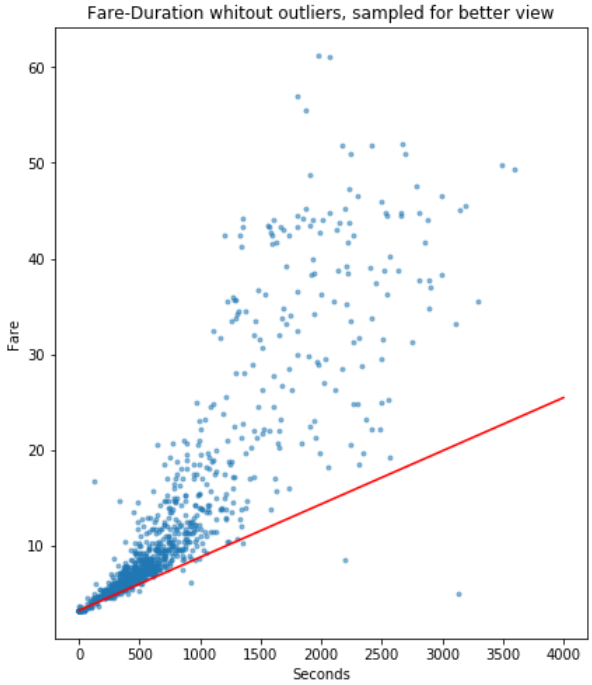


We can see interesting things. There are values in which the fare was hundreds of dollars for only a few miles. These are probably outliers, but we can't assume that they are errors. Maybe some rich person just made the taxi wait. On the other side, trips that were charged 0$ for dozens of miles are clearly errors. We decided to drop these values.

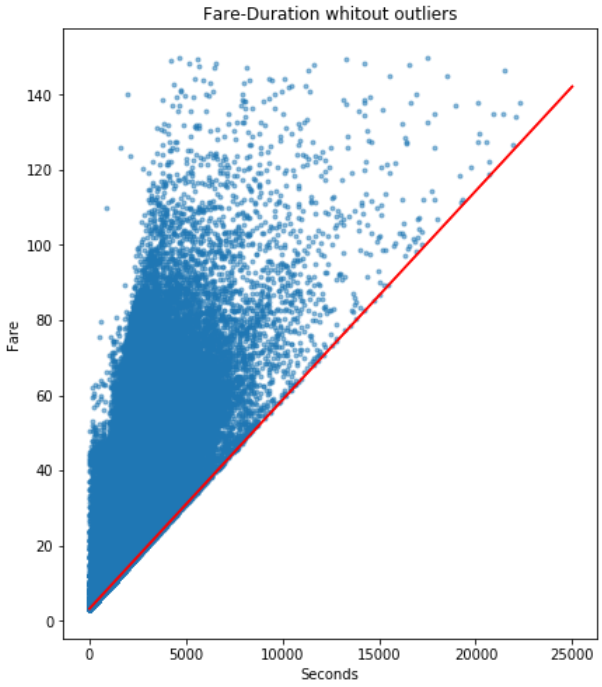
Note that there is a clear linear relationship between these variables.

We proceeded to do the same analysis and cleaning for the fare-duration





Here the relationship is not that clear, but there are clearly weird mistakes too. We decided to drop these weir errors too:



Finally, we decided to do a similar cleaning with the fare-duration relation.

We can assume that a trip with X miles will have at least f(x) seconds, where f(x) is a function that takes the minimum or average speed that a car drives.

To try to avoid eliminating true values, let's assume that the cars go at very high speed (way above average): 50mph. This means that for every mile in the trip, there should be at least 72 seconds of time. To avoid taking into account really short trips, which may not let's also include a minimum miles condition.

Again, we have the following condition:

*IF miles > 5 AND duration > 60*5:*

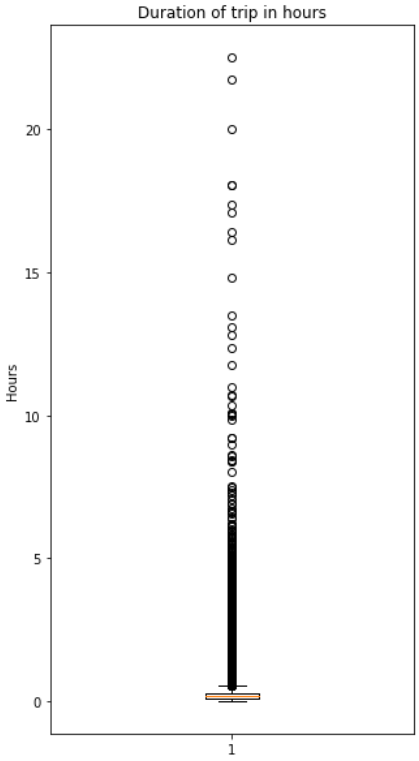
- seconds >= miles * 60

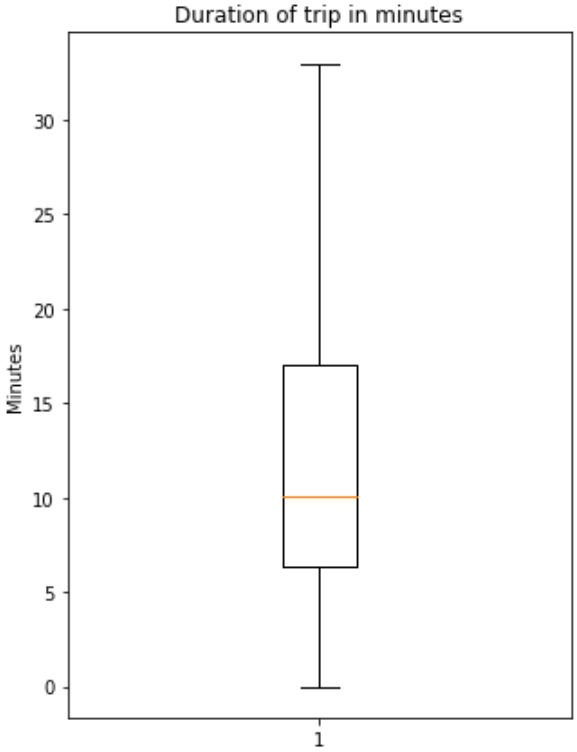
We deleted 0.31% of data that had the condition above.

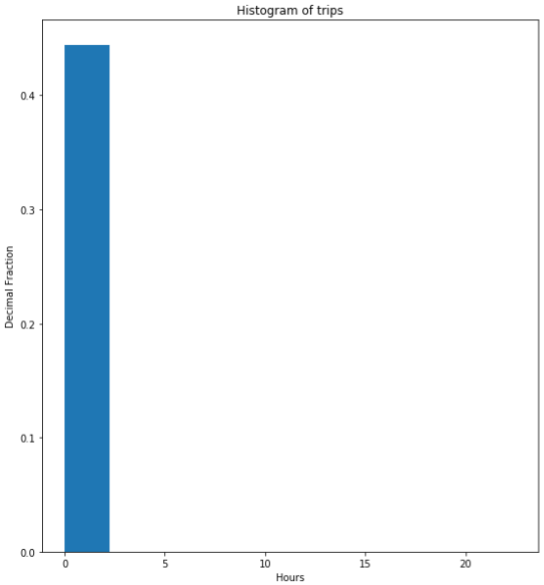
Note: since these three values have a high correlation, we could have made a function to try and estimate the weird values. However, it would have been a difficult task, in which we could have confused outliers with errors. Since we had an enormous quantity of data, we concluded that the easiest was to eliminate them.

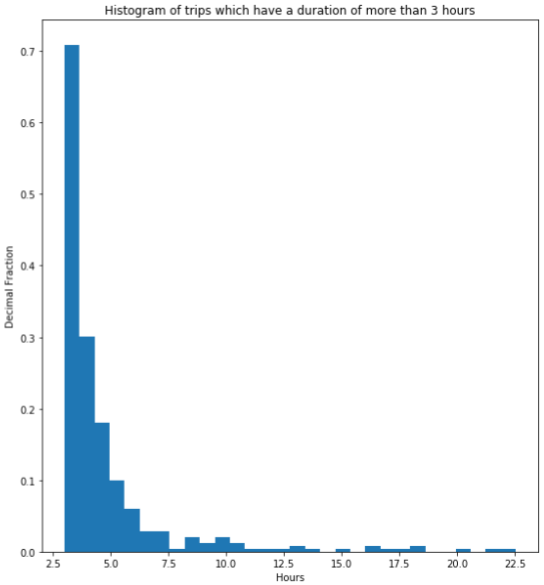
Box-plots: Duration

As we can see, there is an enormous quantity of outliers. We should maybe take them off before feeding them to the models, however, this is something that we will do in the future if necessary.

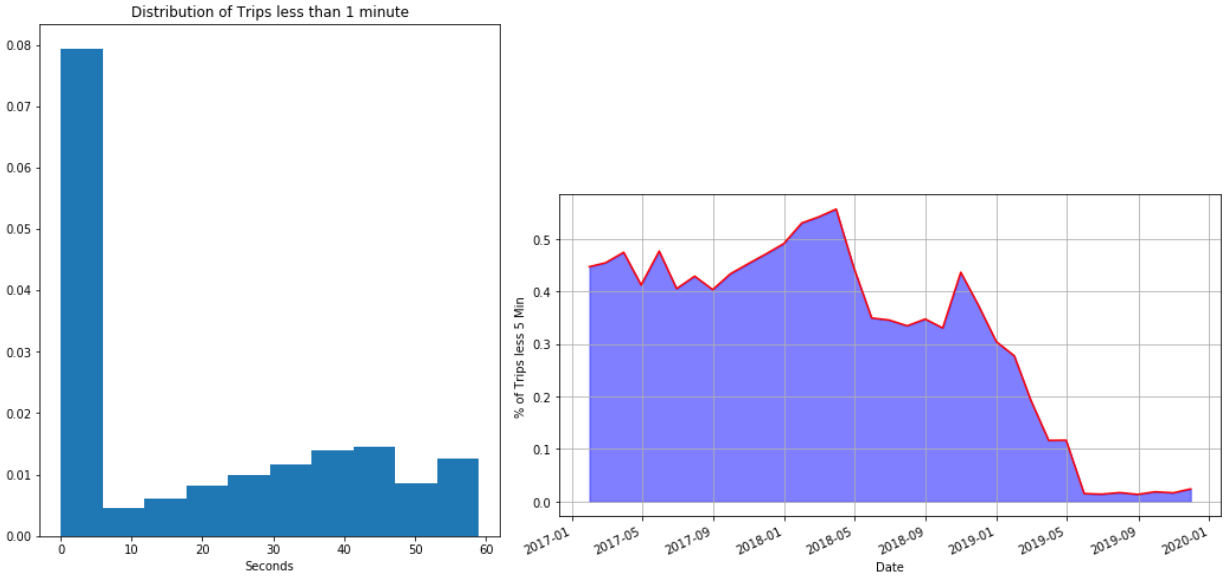






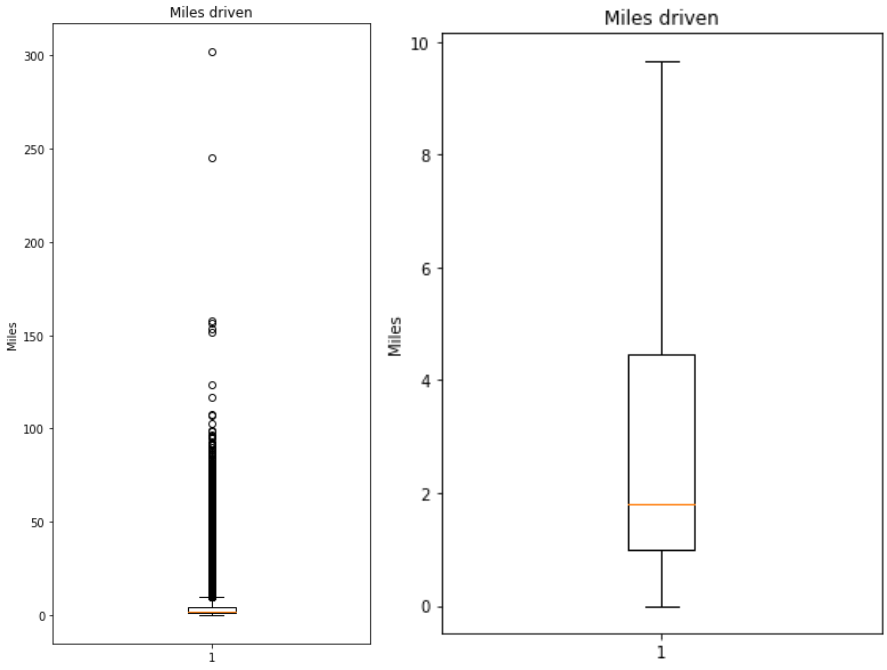


Also, there are a lot of trips that last less than 60 seconds:



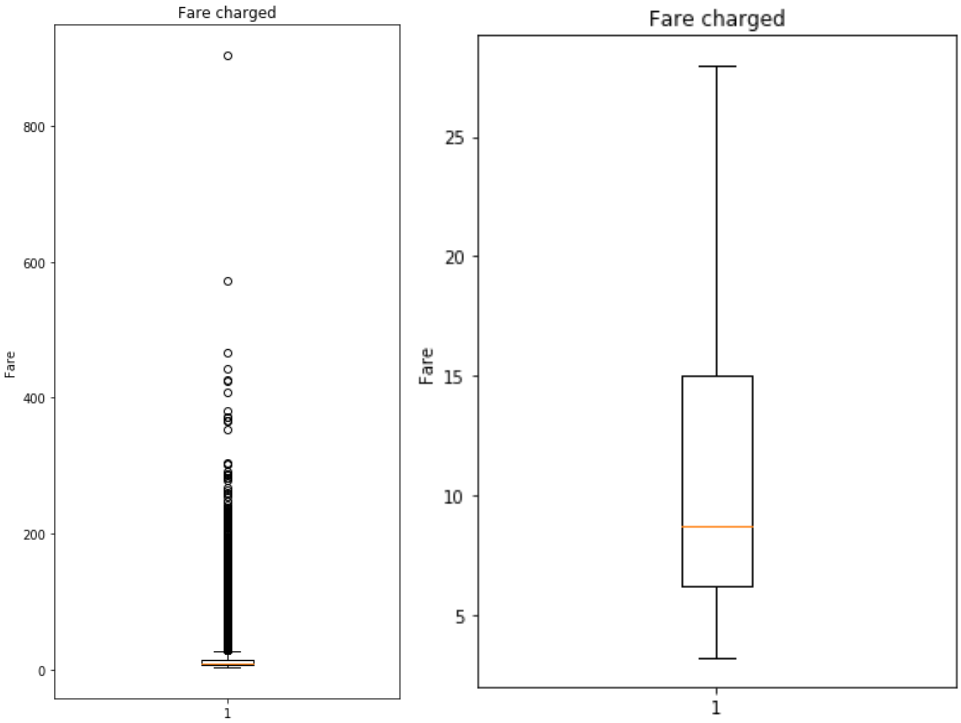
But, these trips are unlikely to be profitable. Are we interested in predicting trips that quick? We should probably take them off too. They also seem to be disappearing in the last year.

Box-plots: Miles



Here we can see there are also many outliers. However, compared to the time, these seem much more concentrated near the top whisker. Trips of more than 50 miles in Chicago don't make much sense.

Box Plots: Fare



Finally and as expected, fare also has a lot of outliers. Once again, later during models tuning we will decide if we take them off or not, depending on the performance increase.

Weather

Variable selection

After inspecting the variables we selected the ones that we considered necessary. Some examples of the variable selection we did:

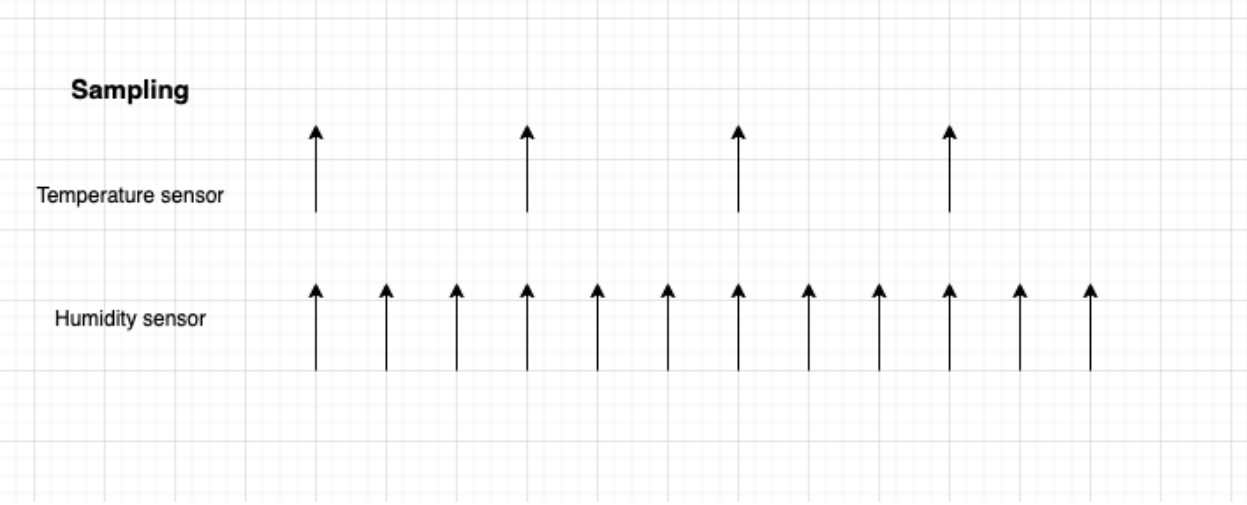
- There were redundant variables such as wind speed in knots and mph. We only selected one of them.
- The station ID, the identification number of the station that collected the data, was useless since we collected the data from only one station.
- Other variables that we considered that had no relationship with predicting taxi trips.

The final variables we ended up having in the weather dataset are:

- **Temperature**: temperature in degrees Fahrenheit, typically at 2 meters.
- **Relative_humidity**: relative humidity
- **Wind_direction**: Wind direction in degrees.
- **Wind_speed**: wind speed in knots
- **Precipitation**: precipitation level
- **sky_level**: categorical variable for whether the day is sunny, cloudy, etc.

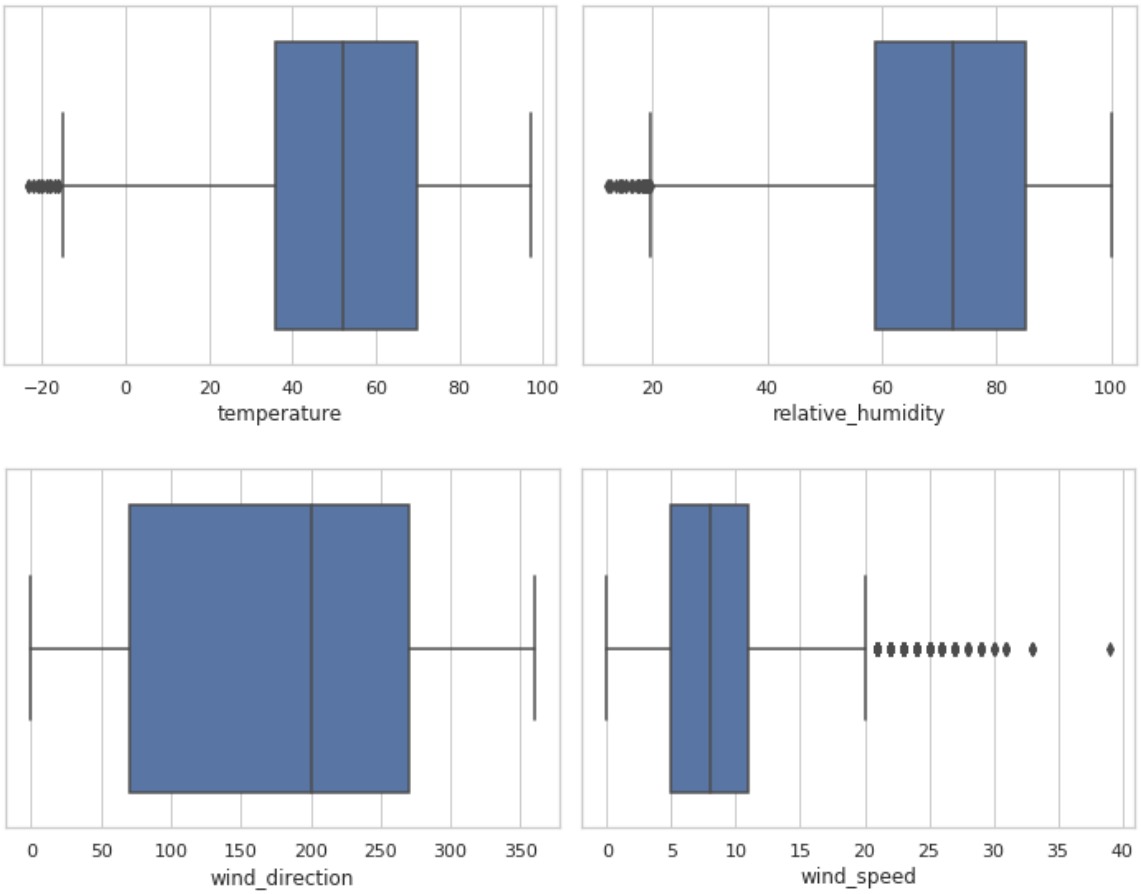
Missing Values

This dataset is in a much better state than the trips dataset. There were a lot of missing values in several variables, but further inspection showed that missing values were due to the join of two dataset with different update frequencies. Since the longest update frequency was still above one hour, we dropped the extra information since we didn't need that much precision.



Errors

We couldn't find any apparent errors. Box-plots were coherent, even if we could find some outliers. Here are some examples:



These outliers may be good for predictions. For example, a temperature of -20 degrees is highly correlated with a poor demand of taxi trips. Therefore we concluded that we should leave them.

Holidays

This was a really simple dataset. Obviously there are no missing values, and the errors were checked manually by inspecting some key values. Indeed, we found that some holidays such as Christmas weren't correctly labelled, so we proceeded to use another source, which had data with no apparent errors.

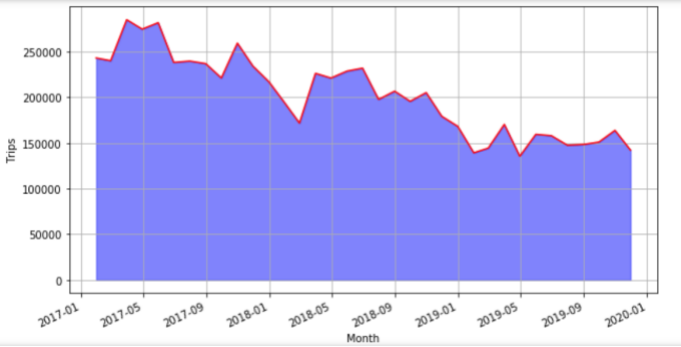
Merging

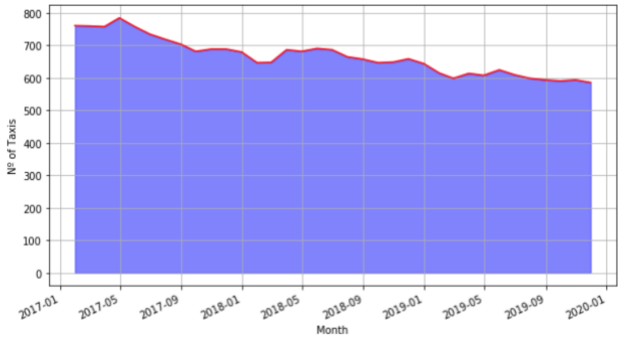
The final task was to merge these three datasets. For that we counted the taxi trips that happened each hour and we grouped them by community. After we added the weather and holidays

characteristics to each hour. Also, some variables were converted depending on the necessity. For example categorical variables were one hot encoded.

Data Analysis

After the Data Cleaning process, we start with the next phase of analysis. First we wanted to know the trend of the sector and therefore we made a temporal analysis of both the demand for trips and the number of taxis per year. These are the results obtained:





As you can see, the demand has been decreasing year after year. It seems reasonable that one of the reasons could be the continuous growth as Uber or Lyft companies that give a similar service to the taxis. The positive point is that it seems that demand has stabilized in the last year (2019), which makes us think that the taxi sector, although it has suffered a fall, will continue to be beneficial in the future.

One of the conclusions we reached after this first analysis was that the year seems to be an important one when it comes to predicting demand, and so we started with the feature engineer phase.

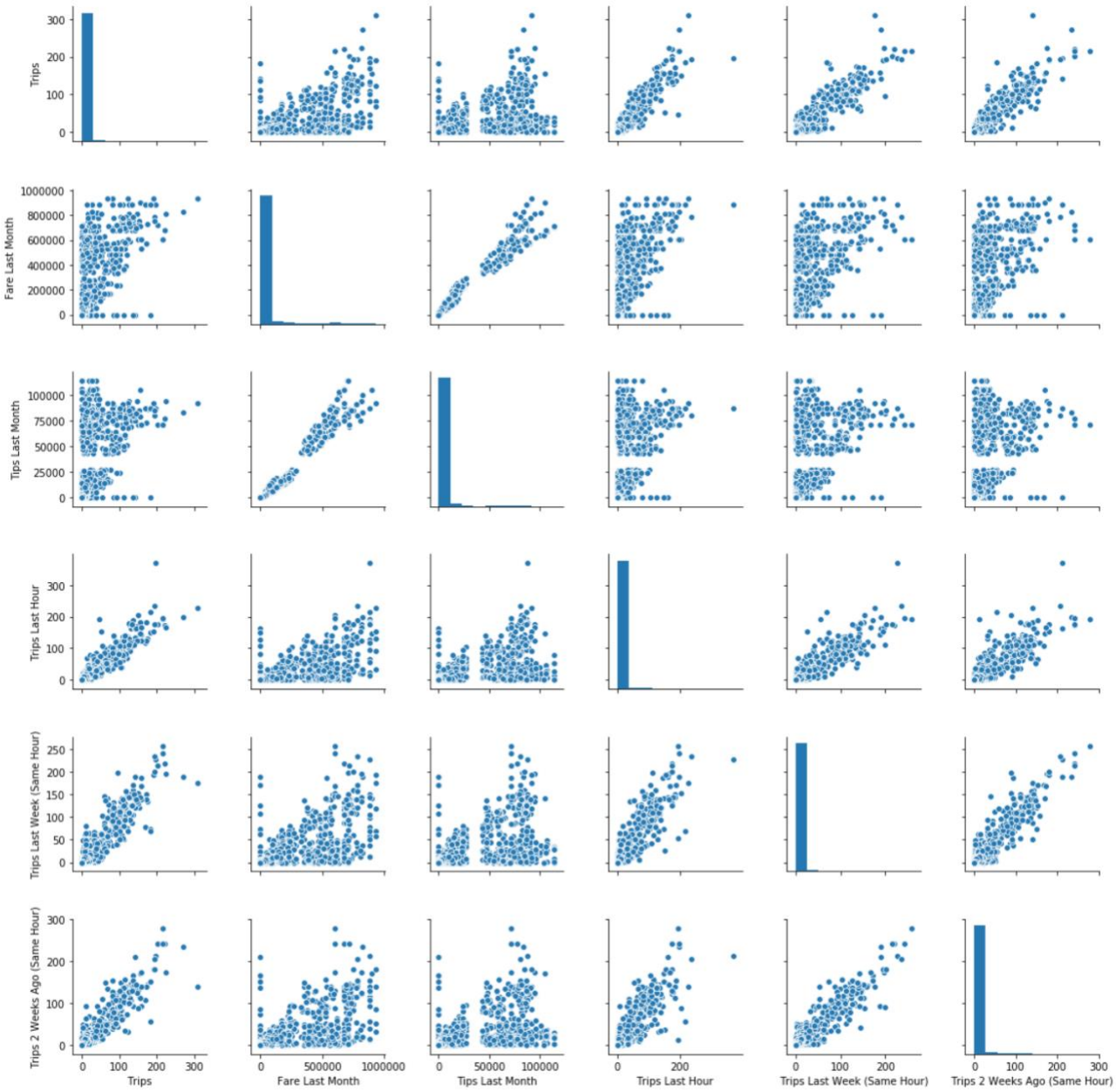
In the following sections, we will analyze the initial features and the features created, grouped in three sets:

Trips Features:

It seems reasonable to have the assumption that the demand that exists at a given time is influenced by the characteristics of previous trips in that particular area or at that particular time. Therefore, we have decided to extract a series of features related to past trips:

- Fare Last Month: total income in that community area in the last month.
- Tips Last Month: total amount of tips in that community area in the last month.
- Trips Last Hour: trips in the last hour in a certain community area.
- Trips Last Week (Same Hour): trips at the same time and day of the week in a certain community area last week.
- Trips 2 Weeks Ago (Same Hour): trips at the same time and day of the week in a certain community area two weeks before.

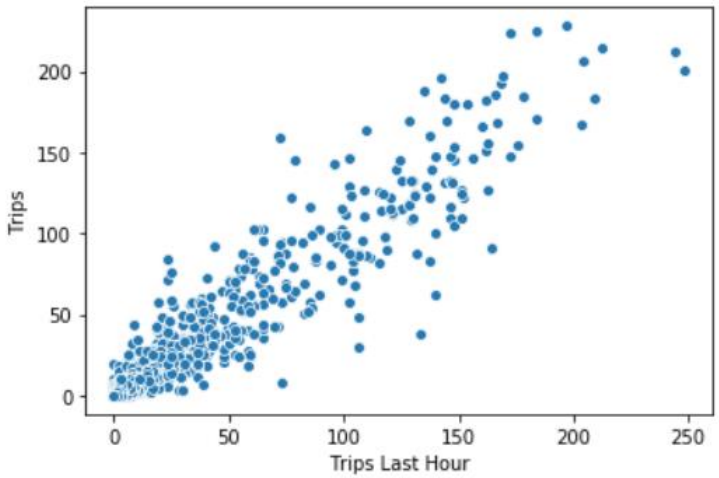
Multicollinearity:



After analyzing the relationship between the variables, we have drawn the following conclusions:

- High correlation between Trips and Demand Features (obviously, these features are equal but just shifted).
- High correlation between Tips Last Month and Fare Last Month.
- There is a gap in medium Tips Last Month Value (between $26 500-$43 000).

In the following figure we show the scatterplot of Trips with Trips Last Hour. We can see how there is a clear relationship between the trips in a certain area and the trips in the previous hour.

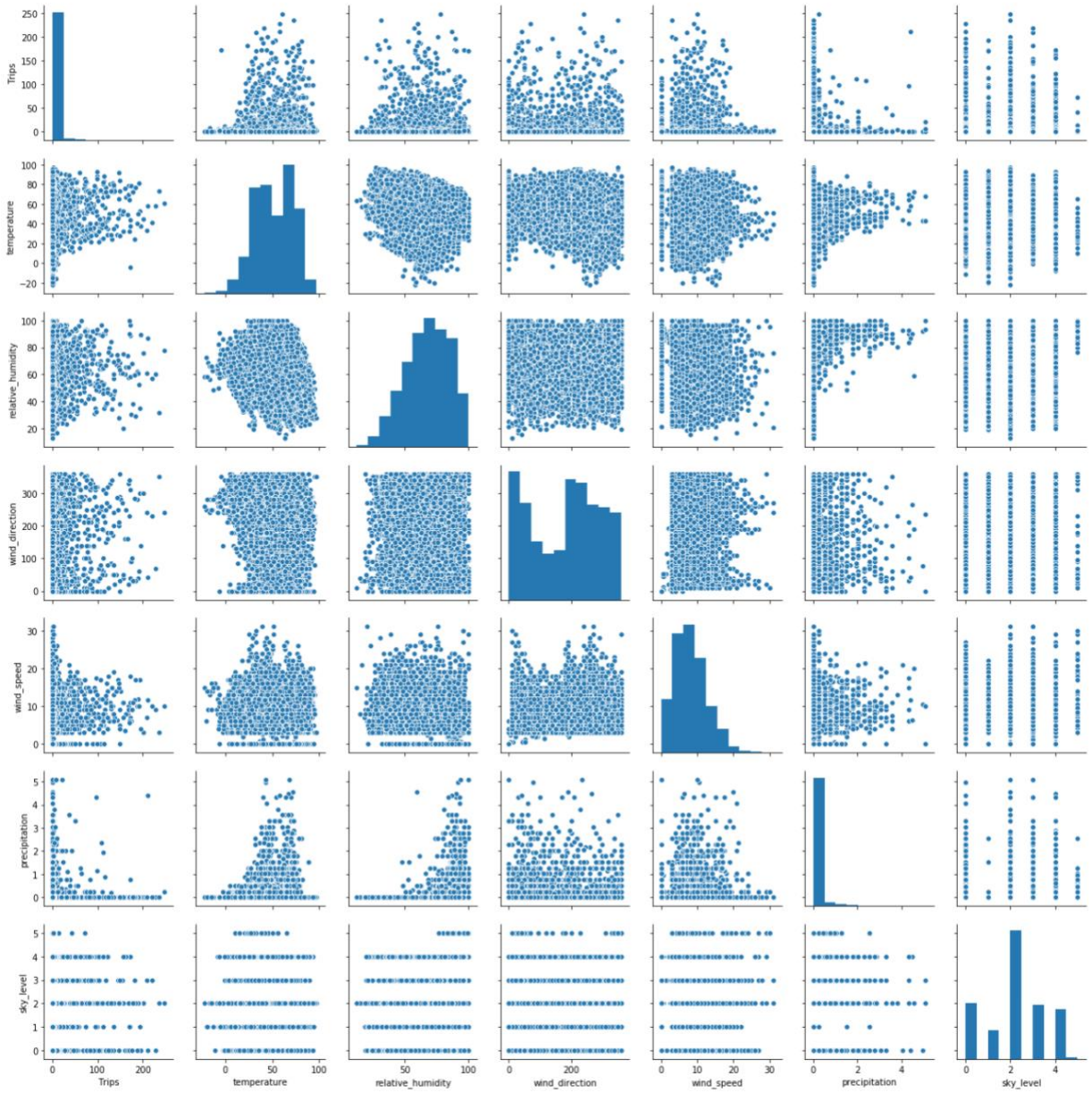


I have also discovered the relationship in Fare and Tips, which seems obvious because the tip is usually a percentage of the fare of the service. However, the gap in the tip values has caught our attention. We have checked if there is a gap in the original dataset, and if so, it was not an error in our data cleaning process. We have not been able to discover the reason for this, since the fact that the variable is the set of Tips in a community area in a month, does not lead us to think that it could be due to a simple coincidence, but it would be necessary to study it more thoroughly in the future. It should also be noted that this relationship seems to be more focused on lower fare amounts, and that it is dispersed as fare increases. The average tip percentage for the whole dataset is 3%.

Weather Features

Multicollinearity

In the following figure we present a pairplot of these variables with the target variable (Trips).

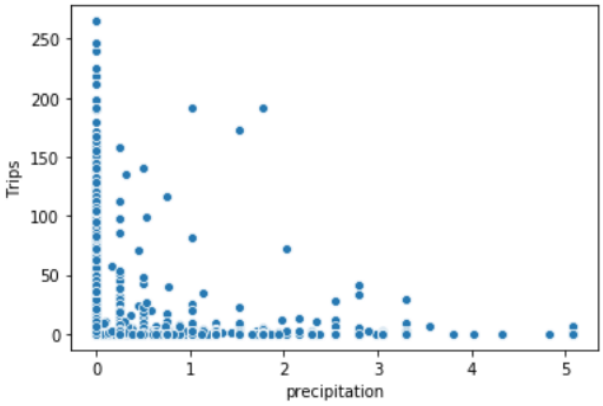


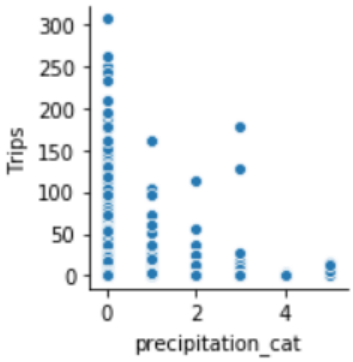
Of the relationships between variables we can highlight:

- Trips with Precipitation: There is a huge difference when precipitation is 0 and when it is not. So we will try to create another variable to check if in a day has rained or not.
- Trips with temperature: There is a difference as well when the temperature is lower than 15ºF.
- Wind variables does not seem to have any relation with the number of trips.

These conclusions lead us to focus on the Precipitation and Temperature variables:

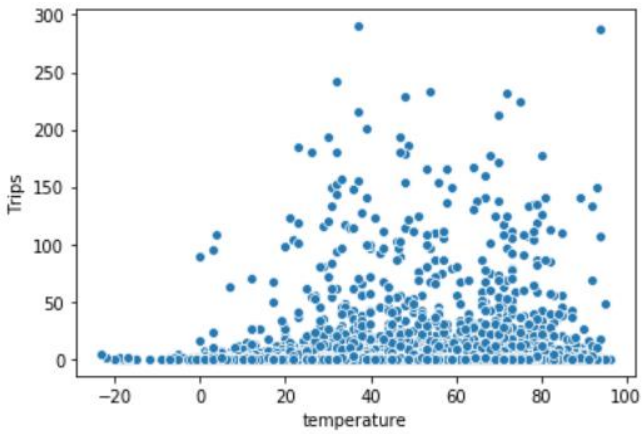
Trips vs. Precipitation





We hypothesize that taking the precipitation variable as an ordinal (with integer values between 0 and 5) rather than as continuous values can better help predict the number of trips. To do this we create the variable **precipitation_cat**, which is obtained by rounding off the precipitation values. To test the hypothesis, we use the ANOVA test for precipitation_cat and the regression test for precipitation, both with Trips. The results are favorable since we obtain a much lower p-value in the case of precipitation_cat, so we consider our hypothesis to be valid (8.38 e-11 for precipitation_cat and 0.87 for precipitation).

Trips vs Temperature



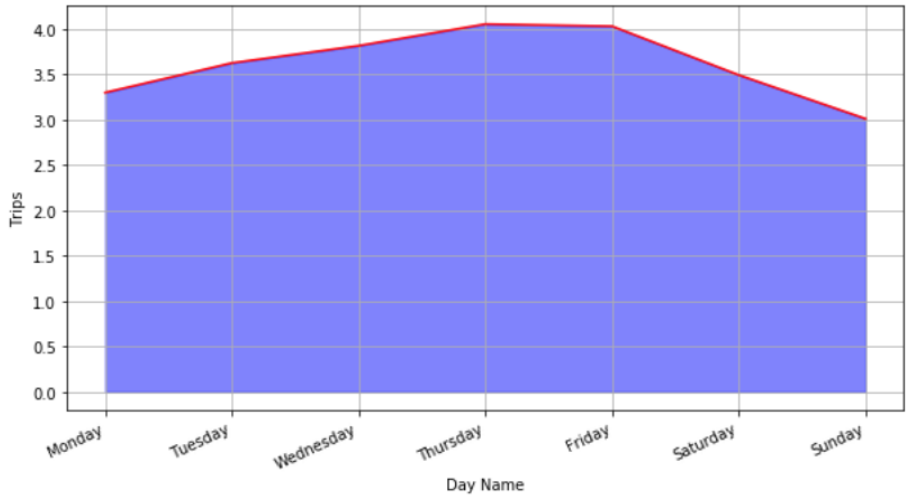
With the temperature variable we have a similar hypothesis to the previous one, we believe that grouping the days by temperature ranges can help to better predict trips, than using the accuracy of using the exact temperature of each day and each hour. In addition, a different distribution of trips seems to be observed as the temperature increases.

Therefore, we decided to create the variable **temperatura_cat** that indicated which of these bins was in that day ([(-30, 0] < (0, 20] < (20, 60] < (60, 100])). To test this hypothesis, we used the same statistical tests as in the previous one (ANOVA and regression). However, this variable obtains a higher p-value in temperature_cat so we decided to keep with the continuous variable.

Date Features

The last group of features we have extracted and analyzed were related to the date of the trips. I have also taken into account the data from the Holiday Dataset resulting in the following predictors:

- Daytype: day type of the trips (W: Workday; A: Saturdays; U: Sunday/public holiday).
- Day Name: day name of the week of the trips.
- Month: month of the trips.
- Hour: hour of the trips.
- Quarter: quarter of year of the trips.
- Year: year of the trips.
- First Half Month: boolean variable that means if the trips are in the first half of the month or in the second one.



Trips vs Day Name

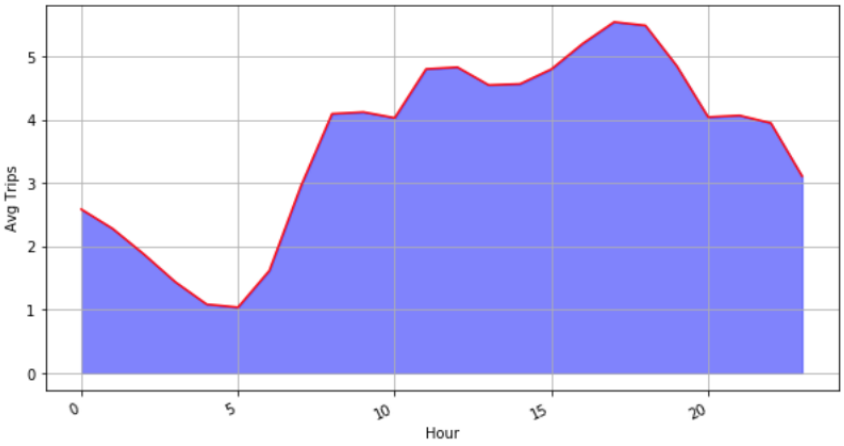
In the below picture it is shown the average trips per hour and per community are in the different day names. It can be seen that Thursday and Friday are the days with the highest demand, while Sunday is the day with the lowest.

Trips vs Day Type

We had the assumption that there would be a considerable difference of the demand between a workday and a public holiday. However, how it is shown in the next picture the average of trips per hour is similar in the three different types. The difference of the average trips between a workday and a public holiday is just 0.25 trips per hour. On Saturdays there is a higher difference but it seems that day type is not the best predictor for estimating the demand of trips.

Trips vs Hour

Logically, the hour at which you want to predict demand is an important factor in that prediction. As can be seen in the following figure there is a large difference in average trips between night and day.



Multicollinearity

Finally, we carried out a study of the correlation between all the final variables of our dataset:

There is a high correlation between:

- Fare Last Month & Tips Last Month.
- Month & Quarter.
- Trips Last Hour & Trips Last Week (Same Hour) &Trips 2 Weeks Ago (Same Hour).
- Day Namber & First Half Month.

These correlations will be taken into account in the design part of the models for the next phase.

Models

In order to send the taxis where the customers will be located, we have to predict the demand of taxis, and therefore we have used several models in order to achieve the best possible accuracy in those predictions.

In order to choose the best possible model, these are the KPIs used:

- Mean Absolute Error (MAE): The absolute of the error
- Mean Square Error (MSE): The mean of the squared error
- Root Mean Square Error (RMSE): The root of the mean square error

The train/test partition has been made according to a real-case scenario, where we have data until one point in time and we have to test our model in the future. In order to do this, we have chosen the last available month in our dataset (November 2019) to be the test set, and all the other months for the training set.

In order to use all the variables appropriately, we have set the following variables to be categorical: pickup_community_area, daytype, sky_level, Day Name, Month, Hour, Year, Quarter. Those categorical variables have been one-hot encoded in order for the models to learn from them. Also, as some of the variables were highly correlated, one of the correlated variables has been dropped for the models where there cannot be correlated variables.

Now we will elaborate the different used models, together with the metrics obtained and the pros and cons of each model.

Linear Regression

- It is one of the simplest and basic models to check for linear relationship among the variables.
- It is used for business models to evaluate trends and make forecasts or predictions and hence it can be utilized in our model for the prediction of taxi trips.

Pros:

- Takes very less time for training the model.
- Easy to use and study.

Cons:

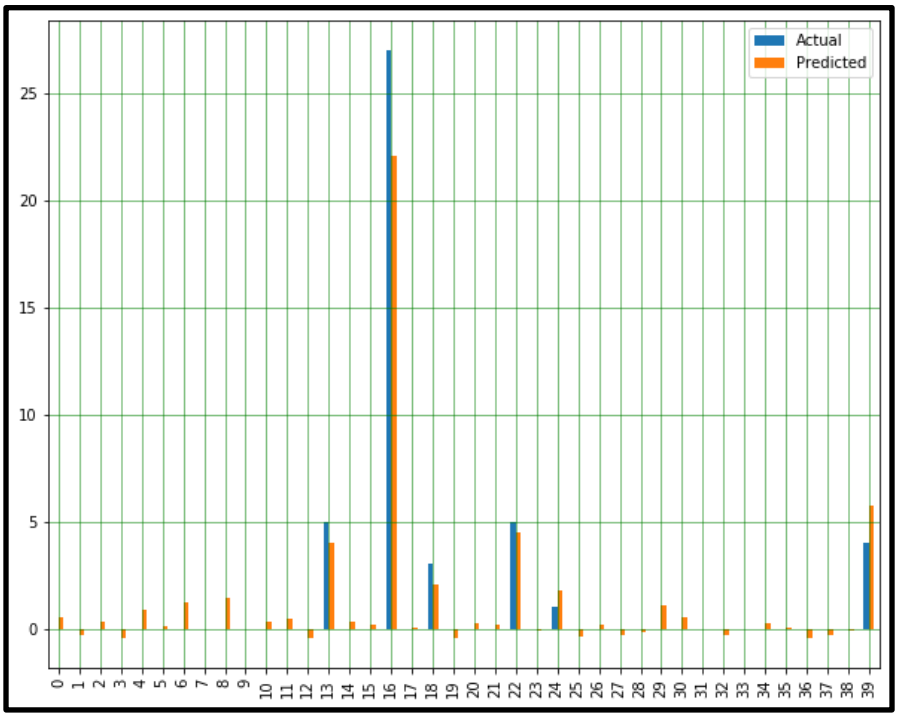
- Highly sensitive to correlation.
- High MAE error

After applying normalization and comparing AIC values of models with some of the insignificant variables dropped, these are the best results of model obtained by OLS regression :

MAE train : 1.1822	**MAE test** : 1.18746
MSE train : 15.6059	**MSE test :** 15.8506
RMSE train : 3.9504	**RMSE test :** 3.9813

Smallest Value of AIC obtained = 5.205e+06

The comparison of the actual taxi trips and predicted taxi trips according to the Linear Regression model is shown below :

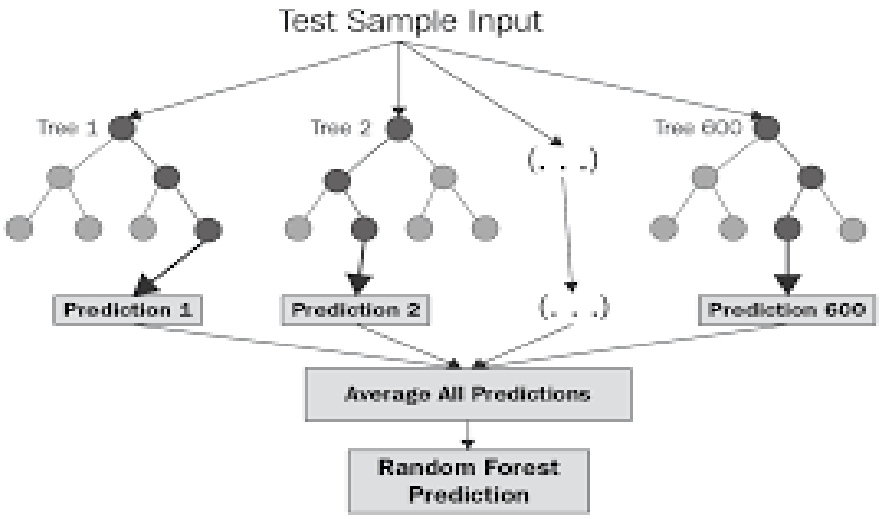


The amount of error is quite visible from the plot. In order to acquire even better results, another model has been used which is Random Forest.

Random Forest

- Random forest is a Supervised Learning algorithm which uses ensemble learning method for classification and regression.

- Random forest is a bagging technique.The trees in random forests are run in parallel. There is no interaction between these trees while building them.
- It operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. A random forest is a meta-estimator (i.e. it combines the result of multiple predictions) which aggregates many decision trees.



Hyperparameters in Random Forest :-

- n_estimators = number of trees in the forest
- max_features = max number of features considered for splitting a node
- max_depth = max number of levels in each decision tree
- min_samples_split = min number of data points placed in a node before the node is split
- min_samples_leaf = min number of data points allowed in a leaf node
- bootstrap = method for sampling data points (with or without replacement), if true then with replacement.
- n_jobs=No. of cores used for training algorithms
- oob_score=whether to use out-of-bag samples to estimate the R^2 on unseen data.

After trying out different hyperparameters it is found out that the hyperparameters used below gave the best metric values.

```
rf4 = RandomForestRegressor(n_estimators = 256,bootstrap=True,min_samples_leaf=64,oob_score=True,n_jobs=-1,
                            max_features=0.4,verbose=4,min_samples_split=64 )
rf4=rf4.fit(x_train, y_train)
```

Results of Random Forest Model :-

MAE train: 0.9796	**MAE test**: 0.9962
MSE train: 12.4537	**MSE test:** 12.707
RMSE train: 3.529	**RMSE test:** 3.5647

Pros:

- It is one of the most accurate learning algorithms available. For many data sets, it produces a **highly accurate classifier**.
- It gives estimates of what variables that are important in the classification.
- It generates an internal **unbiased estimate of the generalization error** as the forest building progresses.

Cons:

- Random forests have been observed to **overfit for some datasets** with noisy classification/regression tasks.
- The main limitation of the Random Forests algorithm is that a large number of trees may make the algorithm slow for real-time prediction.
- For data including categorical variables with different number of levels, random forests are biased in favor of those attributes with more levels. Therefore, the variable importance scores from random forest are not reliable for this type of data.

Gradient Boosting

- "Boosting" in machine learning is a way of combining multiple simple models into a single composite model.
- This is also why boosting is known as an additive model, since simple models (also known as weak learners) are added one at a time, while keeping existing trees in the model unchanged.

- As we combine more and more simple models, the complete final model becomes a stronger predictor. The term "gradient" in "gradient boosting" comes from the fact that the algorithm uses gradient descent to minimize the loss.

Decision trees are used as the weak learners in gradient boosting. Decision Tree solves the problem of machine learning by transforming the data into tree representation. Each internal node of the tree representation denotes an attribute and each leaf node denotes a class label.

The high level steps that we follow to implement Gradient Boosting Regression is as below:

1. Select a weak learner
2. Use an additive model
3. Define a loss function
4. Minimize the loss function

After trying different hyperparameters I found out that the hyperparameters used below gave the best metric values.

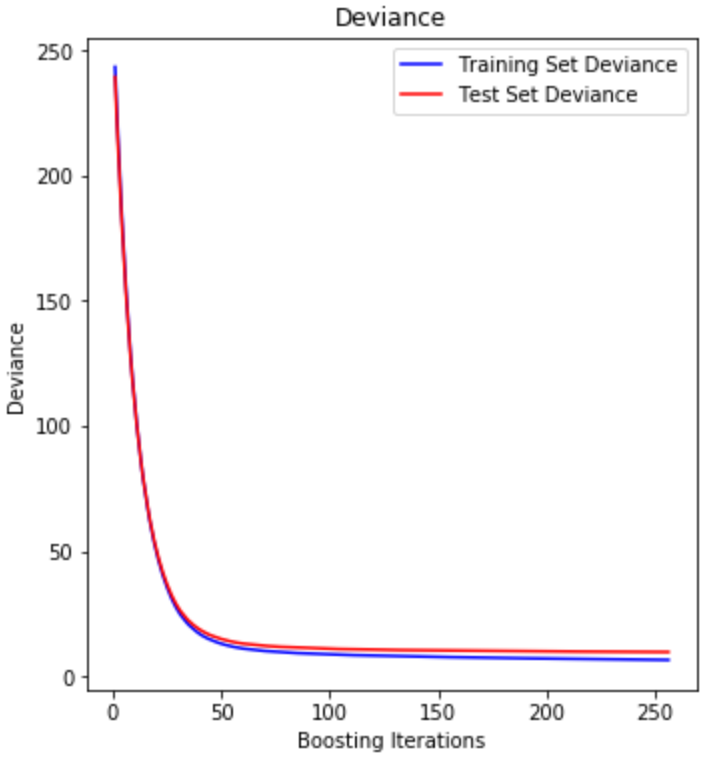
```
params = {'n_estimators': 256, 'max_depth': 16, 'min_samples_split': 64,'min_samples_leaf':32 ,
         'learning_rate': 0.05, 'loss': 'ls','max_features':'sqrt','verbose':4}
clf3 = ensemble.GradientBoostingRegressor(**params)
```

n_estimators - The number of trees in the forest.

criterion : The function to measure the quality of a split. Supported criteria are “mse” for the mean squared error, which is equal to variance reduction as feature selection criterion, and “mae” for the mean absolute error.

max_depth : The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.

min_samples_split : The minimum number of samples required to split an internal node:



Training and validation loss with respect to number of trees

Results of Gradient Boosting :-

MAE train : 0.8157	**MAE test** : 0.9201
MSE train : 6.811	**MSE test :** 10.1646
RMSE train : 2.6098	**RMSE test :** 3.1888

Pros:

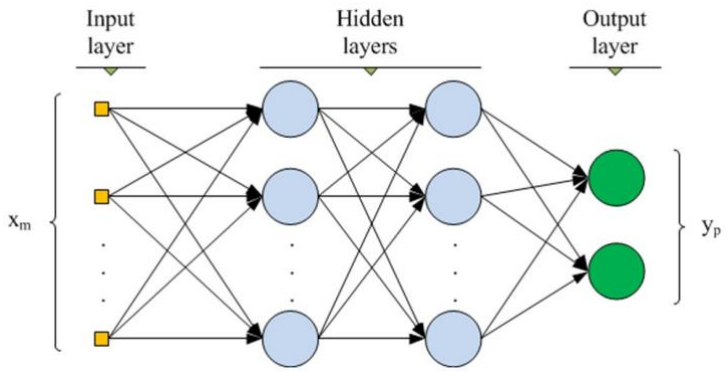
- **Better Accuracy:** Gradient Boosting Regression generally provides better accuracy.
- **Less pre-processing:** Gradient Boosting Regression requires minimal data preprocessing, which helps us in implementing this model faster with less complexity.
- **Higher flexibility:** Gradient Boosting Regression provides can be used with many hyper-parameter and loss functions. This makes the model highly flexible and it can be used to solve a wide variety of problems.

Cons:

- It is sensitive to outliers since every classifier is obliged to fix the errors in the predecessors. Thus, the method is too dependent on outliers.
- The method is almost impossible to scale up. It is because every estimator bases its correctness on the previous predictors, thus making the procedure difficult to streamline.

Neural Networks

Neural networks are the most complex model actually available. Those are used to learn complex nonlinear relationships between variables that other models are unable to learn, therefore we expect this model to find those relationships and behave better than others. The main drawback of this kind of model is that it is a black box, which means that we cannot know how the input is converted into the output.



Our approach has been to use one neural network, which as input has the features and as output the 77 community areas (multiple output regression). In this approach the input features have to be general (of all Chicago), therefore inputs as “Trips in the last hour” have to be averaged, as they cannot be particular per community area.

The hyperparameter tuning has been done with “hypkeras”, a A very simple wrapper for convenient hyperparameter optimization that can be found here:

TODO: reference

https://github.com/maxpumperla/hyperas

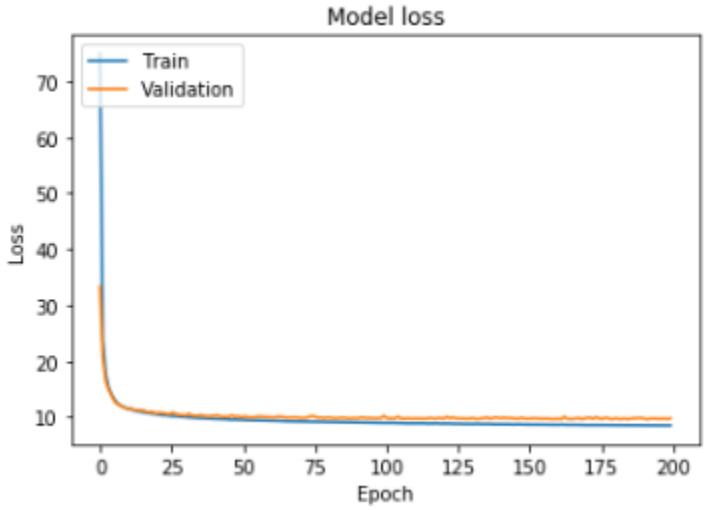
Here we have the tuned hyperparameters, together with their final value:

- Activation function → hyperbolic tangent
- Architecture → Three hidden layers with 32, 64, 32 neurons.
- Optimi161zer → Adam
- Batch size → 64
- Epochs →

Here are the results obtained from the model analysis :

MAE train: 0.973	**MAE test**: 1.288
MSE train: 8.716	**MSE test:** 23.314
RMSE train: 2.952	**RMSE test:** 4.828

As it can be seen, there seems to be a big gap between the training and testing performance. This gap has been tried to be narrowed, by choosing the point with best validation loss and checking the model is not overfitted at this point.



As it can be seen in the picture, the model is not overfitted, but it still behaves far worse in the test set. This could be due to the validation (15% of random data) and test set chosen, their distribution might be different, which could be true as we have chosen the whole last month as a test set.

Pros:

- Best training MSE

Cons:

- Black Box approach
- Worst metrics in test set

Summary

In the next table the summary of all the models can be seen.

	MAE	MSE	RMSE
Linear Regression	1.184	15.39	3.96
Random Forest	0.996	12.706	3.56
Gradient Boosting	**0.907**	**9.699**	**3.114**
Neural Networks	1.288	23.314	4.828

As it can be observed, the best model in all the metrics is the Gradient Boosting algorithm. Therefore, we will use it in order to predict the demand.

Profit

The profit we calculate will be the derived one from the time saved. If our predictions were perfect, by the time a customer asks for a taxi, the taxi would already be there, and therefore the time needed to go from where the taxi was till the customer would be saved, and the customer would be happier too. As our model is not perfect, we will have to take into account the unnecessary movements due to incorrect predictions.

These are the assumptions we are going to make:

- When a taxi leaves a customer in a community, it stays in that community area until another customer asks for it.
- The taxis are constantly in movement, therefore, the cost of gasoline derived from a displacement is null.
- The distance between community areas are calculated taking the middle point of the community area
- Each taxi driver earns 14$/hour

In order to move the taxis we have to know several things:

First we have to calculate the actual disposition of the taxis. This is how it is going to be calculated:

- Actual disposition → Taxis of the previous hour that ended in a community but did not take any other customer.
- Taxis moving → Taxis that are carrying a customer and we know their end point in the next hour.
- Demand prediction → Prediction of the demand at the next hour

With all this information we will try to match the following:

Actual disposition → Demand precision - Taxis moving

We will have to give a command to the taxis that are free to move to the places where it is predicted to be demand, taking into account that some of this demand will be covered by the taxis that are carrying a customer to that community.

Doing this for a given hour, we have moved the taxis obtaining an estimated profit of 228 minutes for an hour, which gets converted into 53$/hour. This, extrapolated to a year makes a profit of 460,000$/year, nearly half a million dollars.

Visualization

Library

The visualization of the results is one of the most important parts of the project. If the client cannot understand and/or use the predictions, all the work was done for nothing.

We considered that an easy way to quickly understand the distribution of the demand of taxi trips in a geographical space, would (logicaly) be a map. Therefore we proceeded to create a mini-library with the help of GeoPandas[6], and the polygons that represent the boundaries of the Chicago community areas[7].

The library, called **mapGeneration** has 2 main functions:

- *loadCommunities*(): which reads and loads the information about the chicago boundaries.
- *mapGenerator*(): function that creates a map showing the predicted taxi trips for each community. It accepts an array or a dictionary.

In addition, there are some options you can pass to the function in order to configure the map:

- *showTaxiTrips*: boolean. If set to true, a label will be superposed in each community with the quantity of taxi trips.
- *saveFig*: string. The path and file name of the result if you want to save it
- *saveByte*: boolean. If set to true, the function will return the image encoded in base64, useful for some applications such as generating it dynamically and showing it in a webpage.
- *legend*: boolean. If set to true the legend of the colors will be shown.
- *cmap*: color map for the community areas. It accepts the same values as matplotlib[8].

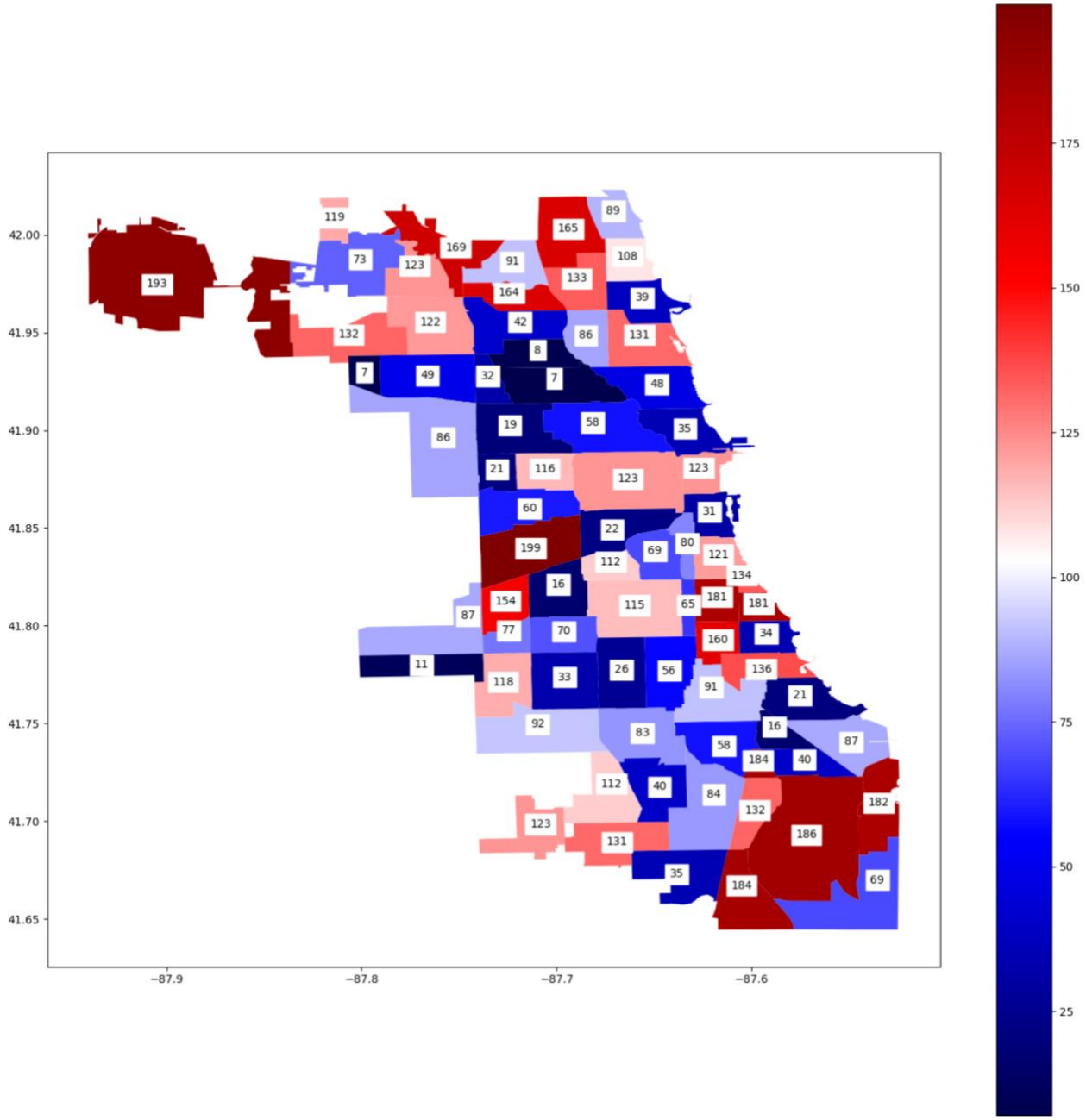
6 Source: https://geopandas.org/

7 Source: https://data.cityofchicago.org/Facilities-Geographic-Boundaries/Boundaries-Community-Areas-current-/cauq-8yn6

8 More information on matplotlib color maps: https://matplotlib.org/tutorials/colors/colormaps.html

Result

Example of generated map with random values:



Deployment

The last step in the project is making it available for production: extracting value from the model predictions.Deployment basically means making the models available to end users of the system, the clients. To do this, we are going to use an API called Flask.

Flask

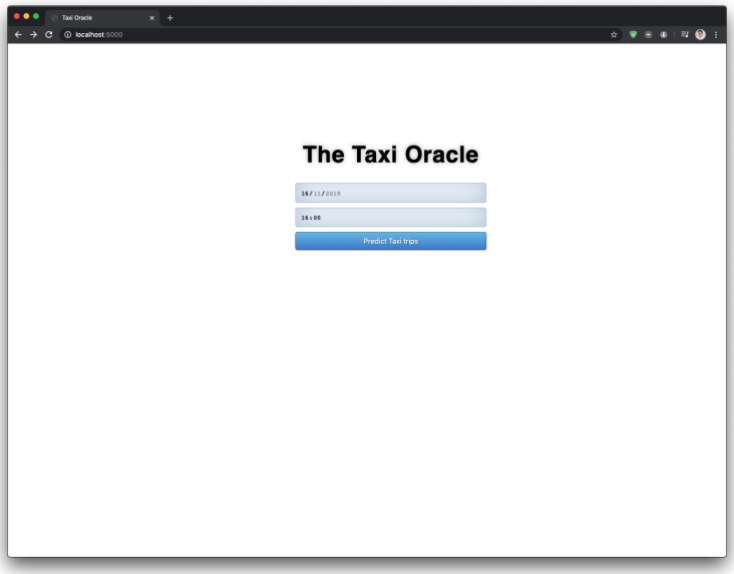
Flask[9] is a microframework, an independent tool that doesn't require any other library for functioning, used for web developing and written in Python. Flask is:

- Easy to use
- Built-in development server and debugger
- RESTful request dispatching
- Extensively documented

The project is composed of three functions which are defined in *app.py*. These functions load models, calculate predictions and deliver results with the help of the *html* and *css* files defined in *static* and *template*.

home()

This function renders *index.html*. Here is where the user inputs the date in which he wants to predict the taxi trips demand. As we can see it's a simple form where a user can input a date and an hour a click on a button to get predictions. The site is simple, but functional.



[9] More information: https://flask.palletsprojects.com/en/1.1.x/

predict()

function that is called when a user asks for predictions on the index and delivers the result as a new html page, *predict.html*.

In the real API the client would only have one map with the best model, but for this specific demonstration we thought it would be better to show the predicted values of the four different models versus the real data.

results()

Finally, we have a function that returns the predictions in a JSON format. This one was not implemented. We just wrote a dummy function for learning purposes and understand how it would work.

Models implementation

Different members of the group were working on different models. To keep work independent, simple and functional, we decided to define a few functions that would help binding the work.

Each model had a folder with it's own binding functions saved in it's own folder. In addition, the models would be saved in a *.pkl* or similar file, so it would easily be loaded by the deployment server. These functions were:

- *binding.TransformDataToX(data)*: a function that makes the necessary transformations to prepare the data for the model input. For example, some models required one-hot encoding.
- *model.predict(X)*: a function that makes predictions given the X matrix.
- *binding.TransformDataToY(result)*: a function that transforms the Y result that the model.predict() function returns and transforms it into an array or a dictionary that the map generator function understands. For example, the model may return the values encoded as a string or floats. This function would convert them to integers.

Lessons Learned

- Performing cleaning and analysis on especially large data
- Models optimization
- Thinking along the sides of the business model to maximize profit
- Time management
- Multitasking
- Working on a collaborative project and using tools such as Github and Basecamp

Possible Next Steps

Change the community area clusters

It could be interesting to change the size of the target variable. For example we could cluster several community areas and end up with less predictors, or on the other side, try to get more precision and dividing them. Would the model be as good?

We could also tune up the model and leave aside community areas that the selected company doesn't work in.

Creating an ensembling method

"Multitudes have unstoppable power". Each model has its own advantages and disadvantages. If we combine the linear regression and random forest we can use their respective heuristic properties to get better results.

Extrapolate to other companies with larger numbers of customers.

The cab company data we used for analysis and training is Flash Cab. The No. of rides by the cab company was less compared to some big company like Uber or Lyft. If we use the same approach with fairly big companies we will be having more rides in each community's area. Which will make the algorithm more robust and profitable.